REMARKS

Claims 1-16 and 20-33 were pending in the application at the time of the Office Action. Claims 1-16 and 20-33 were rejected under 35 U.S.C. 103. By this response, Applicant has amended claims 1, 20, 25-27, 29, 31 and 33. Applicant respectfully submits that the amendment to the claims are based in the specification as originally filed and that no new matter has been added. Entry of the claim amendments is respectfully requested. As such, claims 1-16 and 20-33 are presented for the Examiner's consideration in light of the following remarks.

Reconsideration and allowance of the application is respectfully requested in view of the above amendments to the claims and the following remarks. Applicant requests that the Examiner carefully review any references discussed below to ensure that Applicant's understanding and discussion of the references, if any, is consistent with the Examiner's understanding. For the Examiner's convenience and reference, Applicant's remarks are presented in the order in which the corresponding issues were raised in the Office Action.

A. Examiner Telephone Interview

Applicant(s) and applicant's attorney express appreciation to the Examiner for the courtesies extended during the recent interview held on February 3, 2009. This response includes the substance of the Interview.

B. Rejection on the Merits

1. Rejections under 35 U.S.C. 103

Claims 1-5, 7, 10 12-16, 20-25, 27, 29-32 were rejected under 35 U.S.C. 103(a) over Beisiegel et al. (US Pub No. 2004/0177335 A1), in view of Skrzynski et al. (US Patent No. 6,691,302 B1) and further in view of Fletcher et al. (U.S. Patent No. 7,035,944 B2).

Applicant appreciates the opportunity to explain the invention in person at the Examiner Interview. As discussed, the present invention is related to a service-oriented software development system for developing and implementing service modules. The service-oriented software development system uses system functions to provide its functionality. System functions can include logging, management, memory sharing, caching, etc. At least some of the system functions of the service-oriented software development system are implemented as service-oriented software service modules. In addition, these system functions that are built as

service-oriented software service modules can be run on the same platform that is provided by the service-oriented software development system.

In order to invoke a system function service-oriented software service module, the platform of the service-oriented software development system calls the system function service-oriented software service module and then implements it through its own platform. Thus, the service-oriented software development system leverages its own service-oriented artifacts to perform its own system functionality. *See* Specification, [para 3]. Because having a service-oriented development system that not only creates and manages service, but is itself built on services, this inherently introduces circularity. The specification teaches, with emphasis added:

> [Para 40] Any system or definition that references itself is circular by nature. For the purpose of the present invention, and to address this circularity, we define a portion of the system, hereon referred to as the Kernel, which breaks such circularity in avoiding infinitely recursive definitional references. Furthermore, we define the Kernel module as the core functional coordinator, software service dispatcher, and composition manager of the system of the present invention.

Independent claims 1, 20 and 27 have been amended to clarify that the runtime engine or runtime server includes a Kernel that breaks the circularity created by a service-oriented development system that itself is built on services.

Applicants traverse the Examiner's rejection for obviousness on the grounds that the references – either individually or in combination – fail to teach or suggest each and every element of the rejected claims. By contrast to the presently claimed invention, neither Beisigel nor Skrzynski teach or suggest that the software development system is used to define or consume service modules to perform **its own** system functions and further provides a mechanism for breaking the circularity that such configuration inherently produces as is presently claimed.

Rather, although Beisigel teaches "a service-oriented development model for enterprise applications and an integrated development environment for architecting such applications," (See Para. [0011]), the Examiner states, and Applicant agrees, that Beisigel does not teach or suggest "the system uses software service modules to perform system functions to enable operation of the system itself, wherein execution of the system functions includes the software service modules of the system functions being implemented through the server infrastructure itself." See Office Action, page 3.

Skrzynski addresses the problem where an application uses a service component to access system services of an operating system, but where the service component may be written in a different programming language than the native operating system API. Thus, the interface modules taught in Skrzynski enable service components that would otherwise be incompatible with the native operating system API to be compatible with the native operating system API on which the service component and application are executed. Notably, Skrzynski teaches adding the interface modules onto already-existing service components so that the already-existing service components can communicate with the native operating system API. In other words, Skrzynski does not teach modifying the service components themselves, but rather, adding the interface modules to existing service components to communicate with the native operating system API. There is nothing in Skrzynski that teach or suggests that these service components (e.g., DLLs) are modified in any way to become service-oriented software modules, as that term is understood by those of skill in the art.

Thus, *at most*, Skrzynski teaches that if the "integrated development environment" application taught in Beisigel had service components that were written in a different language than the native operating system API, then interface modules such as those taught in Skrzynski could be used to allow the development environment service components to be operable with the native operating system API when the service component is programmed in a different language than the API.

The Examiner states, and Applicant agrees, that neither Beisiegel nor Skryznski teach or suggest "service interface definitions for the service-oriented software service modules that perform system functions are first described using the set of management and design tools and then consumed by the same set of management and design tools." Office Action, page 4. The Office Action asserted that Fletcher teaches this.

Fletcher teaches using portlets as web service intermediaries. *See* '939 patent, col. 3, ll. 54-55. Fletcher teaches:

> In preferred embodiments, this technique comprises: defining a system interface for a collection of one or more software resources; populating the system interface with one or more management functions; specifying the populated system interface in a service description document; and registering the service description document in a network-accessible registry.

Id. at col. 4, ll. 12-18. System interfaces are further described in Fletcher as:

> A block diagram illustrating a portlet structured as a web service proxy is shown in FIG. 3. As shown therein, portlet proxy 340 includes a deployment interface 310, a system interface 320, and a functional interface 330. The portlet proxy communicates with a portal platform 300 using these interfaces, acting as an intermediary between the portal platform and the software resource 350 which carries out the function of interest . . . .
>
> . . . .
>
> The system interface is used for run-time management of portlets (that is, of web services represented by portlet proxies) by the portal platform. Use of the system interface allows the portal platform to perform functions such as logging of events, billing, and other types of administrative operations pertaining to execution of the web service. This requires 2-way communication between the portal platform and the portlet proxy, and uses novel techniques which are disclosed herein.

Id. at col. 7, ll. 15-45, col. 8, ll. 31-39. Fletcher further goes on to describe the system interface:

> The WSDL document 450 in FIG. 4B defines the system interface, which in the example is named "System" (see element 460). A complex data type named "Event" is defined (see element 470), comprising 2 string parameters and a date parameter. This data type may be used, for example, when exchanging logging data to be recorded in an auditing log file. A "logEvent" operation is defined (see element 490), and in this example is a 1-way operation invoked using a "logEventReceive" message (see element 480) which has a parameter of type Event. In addition, the example defines a "reportUsage" operation which has 2 messages "reportInput" and "reportOutput".

Id. at col. 8, ll. 62-67, col. 9, ll. 1-6. The example of a system interface in Figure 4B shows that the system interface contemplated by Fletcher is a definition for allowing the portal platform and the portlet proxy to communicate. Because Fletcher teaches that the system interface is a "definition" and not a functional interface, Fletcher does not teach "service interface definitions for the service-oriented software service modules that perform system functions are first described using the set of management and design tools and then consumed by the same set of management and design tools."

Significantly, neither Biesiegel, Skryznki, nor Fletcher teach that in order to perform the system functions of the portal platform, that the portal platform calls or invokes web services to perform its own system functions. As discussed above, such a configuration would introduce

circularity, which is addressed in the present invention by a kernel on the run-time server that has a hard-coded portion to break the circularity.

Thus, neither Biesiegel, Skryznki, nor Fletcher teach or suggest:

> a server infrastructure comprising a kernel having a hard-coded portion for getting or fetching service interface definitions….wherein the kernel of the server infrastructure gets or fetches any service interface definitions for the service-oriented software service modules that perform system functions such that the kernel breaks circularity created when the service interface definitions reference the server infrastructure

as recited in independent claim 1;

> a run-time server comprising a kernel that provides a framework utilizing interfaces with pluggable implementations for dispatching the service-oriented software modules, the kernel comprising a hard-coded portion for getting or fetching a definition of any service-oriented software module….wherein the kernel gets or fetches any definition for the service-oriented software service modules that perform system functions such that the kernel breaks circularity created when the definitions reference the run-time server

as recited in independent claim 20; or

> sending the invocation request to a kernel of a run-time server, the kernel having a hard-coded portion for getting or fetching service interface definitions;
>
> at the kernel, receiving the invocation request from the consumer of the service-oriented software service module, the consumer being the service-oriented development system, wherein the kernel of the run-time server gets or fetches any definitions for the service-oriented software service modules that perform system functions such that the kernel breaks circularity created when the definitions reference the run-time server;

as recited in independent claim 27.

In view of Beisiegel, Skrzynski, and Fletcher's failure to teach at least these elements recited in independent claims 1, 20 and/or 27, Applicants submit that the Examiner has failed to set forth a *prima facie* case for obviousness and respectfully request that the rejection be withdrawn.

Dependent claims 2-5, 7, 10, 12-16, 21-25, and 29-32 depend from independent claims 1, 20 and 27 and thus incorporate the limitations thereof. As such, Applicant respectfully submits that claims 2-5, 7, 10, 12-16, 21-25, and 29-32 are distinguishable over the prior art for at least

the same reasons discussed above with respect to claims 1, 20 and/or 27 and request that the obviousness rejection with respect to these claims be withdrawn.

Dependent claims 6, 8 and 9

Claims 6, 8 and 9 were rejected under 35 U.S.C. 103(a) over Beisiegel in view of Skrzynski et al., and further view of Bowman-Amuah (U.S. Pub No. 2001/0052108 A1). Applicant notes that because independent claim 1, from which claims 6, 8 and 9 depend was rejected under a combination of Beisiegel, Skrzynski and Fletcher that the Office Action inadvertently left Fletcher off of the rejection.

Dependent claims 6, 8 and 9 depend from independent claim 1 and thus incorporate the elements thereof. As such, Applicant respectfully submits that claims 6, 8 and 9 are distinguishable over the prior art for at least the same reasons discussed above with respect to claim 1. Furthermore, the development architecture framework taught by Bowman-Amuah does not cure the deficiencies of Beisiegel/Skrzynki/Fletcher. As such, Applicant requests that the obviousness rejection with respect to these claims be withdrawn.

Dependent claims 11, 26, 28 and 33

Claims 11, 26, 28 and 33 were rejected under 35 U.S.C. 103(a) over Beisiegel et al. in view of Skrzynski et al. and further view of Lai (U.S. Pub No. 2005/0044197 A1). Applicant notes that because independent claim 1, 20 and 27, from which claims 11, 26, 28 and 33 depend was rejected under a combination of Beisiegel, Skrzynski and Fletcher that the Office Action inadvertently left Fletcher off of the rejection.

Dependent claims 11, 26, 28 and 33 depend from independent claims 1, 20 and/or 27 and thus incorporate the elements thereof. As such, Applicant respectfully submits that claims 11, 26, 28 and 33 are distinguishable over the prior art for at least the same reasons discussed above with respect to claims 1, 20 and/or 27. Furthermore, the Web Services architecture taught by Lai does not cure the deficiencies of Beisiegel/Skrzynki/Fletcher. As such, Applicant requests that the obviousness rejection with respect to these claims be withdrawn.

C. Conclusion

In view of the foregoing, and consistent with the tentative agreement reached during the Examiner Interview, Applicants believe the claims as amended are in allowable form.

Applicant notes that this response does not discuss every reason why the presented claims are distinguished over the cited prior art. Most notably, applicant submits that many if not all of the dependent claims are independently distinguishable over the cited prior art. Applicant has merely submitted those arguments which it considers sufficient to clearly distinguish the claims over the cited prior art.

In the event that the Examiner finds remaining impediment to a prompt allowance of this application that may be clarified through a telephone interview, the Examiner is requested to contact the undersigned attorney.

The Commissioner is hereby authorized to charge payment of any of the following fees that may be applicable to this communication, or credit any overpayment, to Deposit Account No. 23-3178: (1) any filing fees required under 37 CFR § 1.16; (2) any patent application and reexamination processing fees under 37 CFR § 1.17; and/or (3) any post issuance fees under 37 CFR § 1.20. In addition, if any additional extension of time is required, which has not otherwise been requested, please consider this a petition therefor and charge any additional fees that may be required to Deposit Account No. 23-3178.

Dated this 11th day of February, 2009.

Respectfully submitted,

/sara d. jones/ Registration # 47,691
SARA D. JONES
Registration No. 47,691
Attorney for Applicant
Customer No. 022913

SDJ: vlr